Effektives Programmieren (6)

Sortieren mit dem Computer — (Teil 3)

Nachdem wir in den letzten beiden Folgen mehrere einfache Sortieralgorithmen besprochen haben, wollen wir uns diesmal nur mit einem einzigen Algorithmus auseinandersetzen, dem Heapsort. Heapsort ist mit Sicherheit die komplizierteste der bekannten Sortiermethoden und verdient deshalb unsere ganze Aufmerksamkeit, zumal sie dazu noch eine der schnellsten ist.

Zuerst einmal zum Ausdruck »heap«. Ein Heap (engl.: Haufen, Menge) bezeichnet ein Variablenfeld, das auf eine ganz bestimmte Art sortiert wurde und damit einige wichtige Eigenschaften erfüllt, auf die wir später noch zu sprechen kommen werden.

Grundlagen

Beschäftigen wir uns zunächst einmal mit dem Grundgedanken des Heapsort. Unsere bisherigen Sortierprogramme waren alle relativ langsam. In den beschriebenen Formeln konnte man das durch den Faktor A^2 erkennen, der besagt, daß die Sortierzeit mit dem Quadrat der Feldgröße steigt. Die Ursache für diese ungünstigen Zeiten war die oft unnötige Anzahl von Durchläufen, obwohl gewisse Voraussetzungen schon gegeben waren. Anders ausgedrückt haben unsere bisherigen Programme bei ihren Durchläufen nie alle Informationen genutzt, die ihnen zur Verfügung standen. Heapsort hat an dieser Stelle etwas geändert. Der Sortieralgorithmus verarbeitet bei jedem Durchlauf mehrere Informationseinheiten, die dann ein systematischeres Vorgehen erlauben.

Was ist ein »Baum«?

Um Heapsort zu verstehen, müssen wir uns jetzt ersteinmal mit dem mathematischen Begriff des »Baums« beschäftigen. Ein mathematischer Baum ist eine Anordnung von Elementen einer Menge. Ein Beispiel für einen Baum sehen Sie in Bild 1. Jetzt stellt sich natürlich die Frage, wie wir einen solchen Baum in unsere Feldstruktur überführen können, denn immerhin ist ein Baum zweidimensional, wir arbeiten jedoch nur mit eindimensionalen Feldern. Sehen Sie sich dazu einmal Bild 2 an. Es zeigt den Baum aus Bild 3 in eindimensionaler Darstellung als Feld. Wie Sie sehen können, ist die Aufstellung streng hierarchisch und berechnet sich wie folgt: Wir nehmen ein Element an einer Position I. Handelt es sich bei unserer Menge aus Elementen um einen Heap (Variablenfeld), so muß der Wert an der Stelle INT (I/2) größer oder gleich des Wertes an der Stelle I sein. Mathematisch beschrieben sieht die Sache folgendermaßen aus (wobei A$(I) ein Feldelement ist):

A$(INT(I/2)) > = A$(I), für 2 < = I < = A (A ist die Gesamtanzahl der Elemente des Feldes).

Anschaulich können wir sagen, daß das Element A$(INT(I/2)) immer der nächsthöhere »Knoten« des Baumes über den Elementen A$(I) und A$(I + 1) ist.

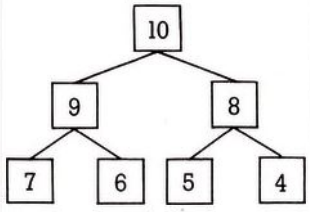

Bild 1. Ein kleines Beispiel einer Baumstruktur

Feld: 10,9,7,5,4,3,2

Bild 2. Die Elemente aus Bild 3 stehen so im Feld

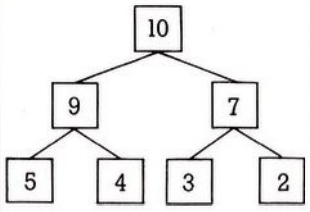

Bild 3. Beispiel für einen Heap (Heap = Haufen, Menge)

Welchen Sinn hat die Baumstruktur?

Sie werden sich jetzt sicherlich fragen, was denn der Vorteil dieser recht komplizierten Baumstruktur ist. Nun, wir sind jetzt in der Lage, in einem Feld, welches einer Baumstruktur entspricht, ganz einfach nach einem Element die Position der nächstgrößeren, beziehungsweise die Position der nächstkleineren Elemente festzustellen und haben damit schon einmal eine Vorsortierung erreicht. Wie Sie vielleicht schon gemerkt haben, lassen sich die Positionen durch die oben genannte Formel ganz leicht aus der Position des Ausgangselements berechnen. Wenn Sie sich noch einmal die Bedingung für einen Baum betrachten, so werden Sie erkennen, daß auch Sonderfälle auftreten können. So kann ein Feld beispielsweise keinen vollständigen Baum darstellen, sondern nur ab einer gewissen Stelle eine Sortierung aufweisen, die der obigen Formel entspricht (Bild 3 zeigt ein Beispiel für eine solche Sortierung). Diese Form der Sortierung bezeichnet man als Heap. Ein Heap ist also nicht vollständig durchsortiert; er zeigt jedoch eine Eigenschaft, die für unser Sortieren von entscheidender Bedeutung ist: das Maximum des Feldes steht jeweils an der Spitze, also auf Position 1.

Nehmen wir jetzt einmal an, Sie entfernen das Element an der Spitze des Heap und setzen den Rest des Feldes wieder zusammen. Der Arrayrest besteht jetzt aus A-1 Elementen, unter denen sich nun das zweitgrößte befindet. Jetzt stellen Sie diese A-1 Elemente wieder so zusammen, daß ein Heap entsteht...

Wie die Reihe weitergeht, können Sie sich jetzt bestimmt denken. Wir nehmen wiederum das erste Element weg und haben somit das insgesamt zweitgrößte Element aussortiert. Diese Schritte wiederholen sich so lange, bis nur noch ein Element, das kleinste, im Heap übrigbleibt. Durch die relativ geordnete Struktur des Variablenfeldes während der Arbeit sind keine langen Suchzeiten erforderlich. Das verkürzt die Sortierdauer erheblich.

Die Formel zeigt die Berechnung der mittleren Anzahl der Bewegungen bei einem Sortiervorgang, wobei A die Anzahl der Elemente des Feldes angibt:

mittbew = ungefähr 1/2*A*ld(A)

Heapsort ist schneller, als die bisherigen Sortiermethoden

An dieser Stelle zeigt sich bereits der deutliche Unterschied von Heapsort zu den vorher besprochenen Sortiermethoden. Während diese immer den Faktor A^2 in ihren Formeln hatten, steigt die Zahl der Bewegungen bei Heapsort im Mittel nur um den Faktor ld (A), wobei ld den Logarithmus zur Basis 2 darstellt. Der Logarithmus ld kann durch den natürlichen Logarithmus (ln) auf die folgende Art dargestellt werden:

ld(A) = ln(A)/ln(2).

An dieser Stelle gleich einmal eine Bemerkung zum Logarithmus: Auf dem C 64 und auf dem VC 20 existiert jeweils nur der natürliche Logarithmus (ln). Im Basic-Dialekt dieser Computer wird jedoch eine unübliche Abkürzung für diesen Logarithmus verwendet. Eine Berechnung erfolgt hier mit dem Befehl LOG, der eigentlich für den Logarithmus der Basis 10 (Zehnerlogarithmus) steht.

Jetzt aber wieder zurück zu Heapsort. In Bild 4 können Sie erkennen, was vorher besprochen wurde. Unser Beispiel-Heap aus Bild 3 wird sortiert, wobei jeweils das erste Element aussortiert und dann der Rest des Arrays wieder zusammengesetzt wird.

Das Programm zu Heapsort ist in Listing 1 dargestellt und benötigt zu seiner Ausführung die beiden anderen Programmteile,

die als Listing 1 und 2 in der 64'er, Ausgabe 4/1985 auf Seite 149 abgedruckt wurden. Um die Abfragepunkte bei Heapsort deutlich darzustellen, wurde in Listing 1 auf FOR-NEXT-Schleifen verzichtet, was allerdings zu Lasten der Geschwindigkeit geht. Wenn wir in der nächsten Ausgabe Quicksort besprechen, werden wir unter anderem auch auf Geschwindigkeitsfaktoren (Garbage-Collection, Vergleiche, Indizierung) eingehen. Außerdem werden wir einen Zeitvergleich über alle bisher besprochenen Routinen durchführen, zu dessen Zweck die Programme natürlich optimiert werden. Bild 5 zeigt einen Ausdruck von Heapsort während der Arbeit. In Bild 6 sehen Sie, wie schon in den vorherigen Folgen, einen Programmablaufplan, anhand dessen es keine großen Probleme bereiten sollte, Heapsort für eigene Anwendungen oder auf andere Programmiersprachen umzuschreiben.

Hiermit wollen wir es für heute aber auch schon bewenden lassen. Heapsort dürfte Sie sicherlich noch einige Zeit beschäftigen, da diese Methode nicht ganz leicht zu durchschauen ist. In der nächsten Folge werden wir uns ausführlich mit dem Quicksort beschäftigen, dem wohl schnellsten Sortieralgorithmus.

(Karsten Schramm/gk)

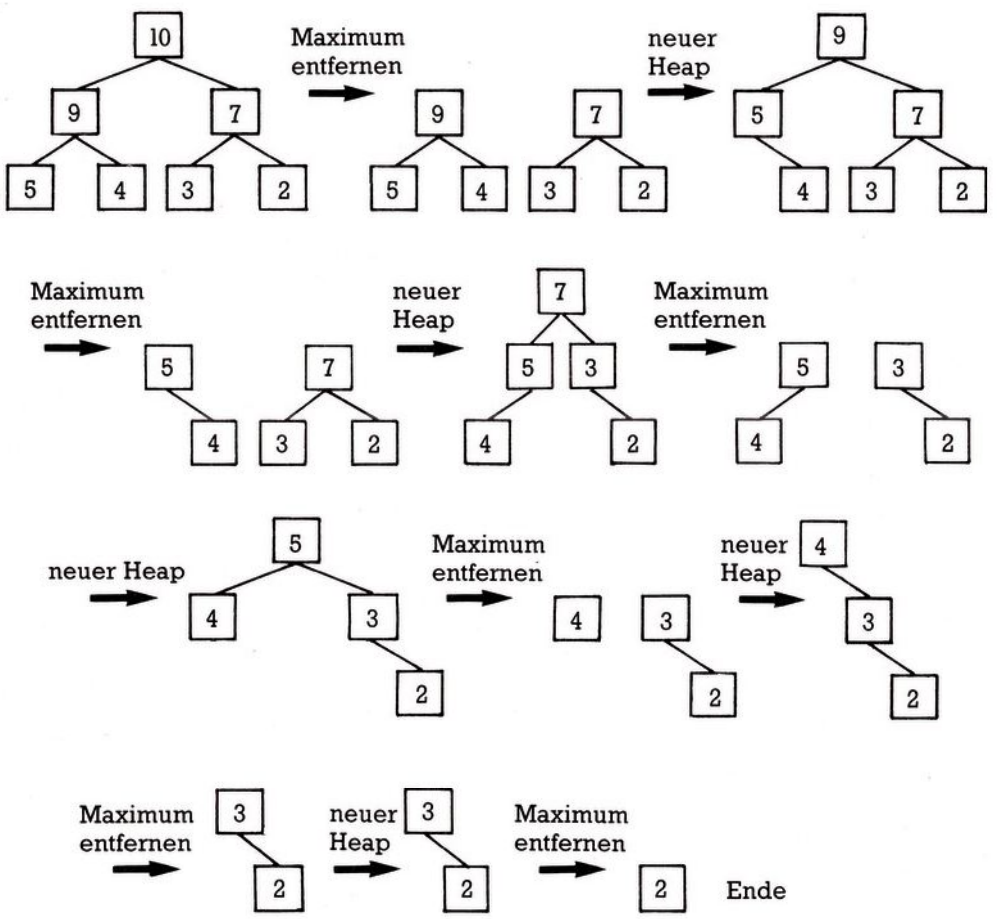

Bild 4. So funktioniert im Prinzip der Heapsort

```
WJH AMJ KPB MRB ETQ KOG MOO TFJ JEE UCV
WJH AMJ KPB MRB UCV KOG MOO TFJ JEE ETQ
WJH AMJ KPB TFJ UCV KOG MOO MRB JEE ETQ
WJH AMJ MOO TFJ UCV KOG KPB MRB JEE ETQ
WJH UCV MOO TFJ ETQ KOG KPB MRB JEE AMJ
WJH UCV MOO TFJ ETQ KOG KPB MRB JEE AMJ
UCV TFJ MOO MRB ETQ KOG KPB AMJ JEE WJH
TFJ MRB MOO JEE ETQ KOG KPB AMJ UCV WJH
MRB JEE MOO AMJ ETQ KOG KPB TFJ UCV WJH
MOO JEE KPB AMJ ETQ KOG MRB TFJ UCV WJH
KPB JEE KOG AMJ ETQ MOO MRB TFJ UCV WJH
KOG JEE ETQ AMJ KPB MOO MRB TFJ UCV WJH
JEE AMJ ETQ KOG KPB MOO MRB TFJ UCV WJH
ETQ AMJ JEE KOG KPB MOO MRB TFJ UCV WJH
AMJ ETQ JEE KOG KPB MOO MRB TFJ UCV WJH

AMJ ETQ JEE KOG KPB MOO MRB TFJ UCV WJH
 10  ELEMENTE
```

Bild 5. Ein Feld wird im Heapsort sortiert. Man erkennt, daß sich die Geschwindigkeit des Heapsort erst bei größeren Feldern bemerkbar macht.

```
10000 REM SORTIEREN MIT BAEUMEN                    <104>
10010 REM                                          <208>
10020 REM HEAPSORT                                 <064>
10030 REM                                          <228>
10040 LG=INT(A/2)+1:RG=A                           <107>
10050 IF RG<=1 THEN 10190                          <197>
10060 IF LG<=1 THEN 10100                          <192>
10070 REM AUFBAU DES HAUFENS                       <166>
10080 LG=LG-1                                      <059>
10090 I=LG:GOTO 10140                              <216>
10100 REM WEGNEHMEN DES MAXIMUMS                   <021>
10110 S$=A$(1):A$(1)=A$(RG):A$(RG)=S$              <137>
10120 RG=RG-1                                      <111>
10130 I=1                                          <229>
10140 X$=A$(I)                                     <240>
10150 P=0: REM FLAG FUER NICHT GEFUNDEN            <214>
10160 IF 2*I<=RG AND P=0 THEN 10200                <002>
10170 A$(I)=X$                                     <014>
10180 GOSUB 3000: GOTO 10050                       <244>
10190 GOTO 10300: REM ENDE                         <087>
10200 REM FELDVARIABLE A$(I) EINORDNEN             <144>
10210 J=2*I                                        <044>
10220 IF J<RG THEN IF A$(J)<A$(J+1)THEN J=
      J+1                                          <188>
10230 IF X$>=A$(J)THEN 10260                       <039>
10240 A$(I)=A$(J)                                  <217>
10250 I=J:GOTO 10160                               <050>
10260 P=1: REM FLAG FUER PLATZ GEFUNDEN            <091>
10270 GOTO 10160                                   <199>
10280 :                                            <138>
10290 :                                            <148>
10300 REM ENDE                                     <015>
```

Listing 1. Das Programm zum Heapsort (Sortieren mit Baumstruktur) ►

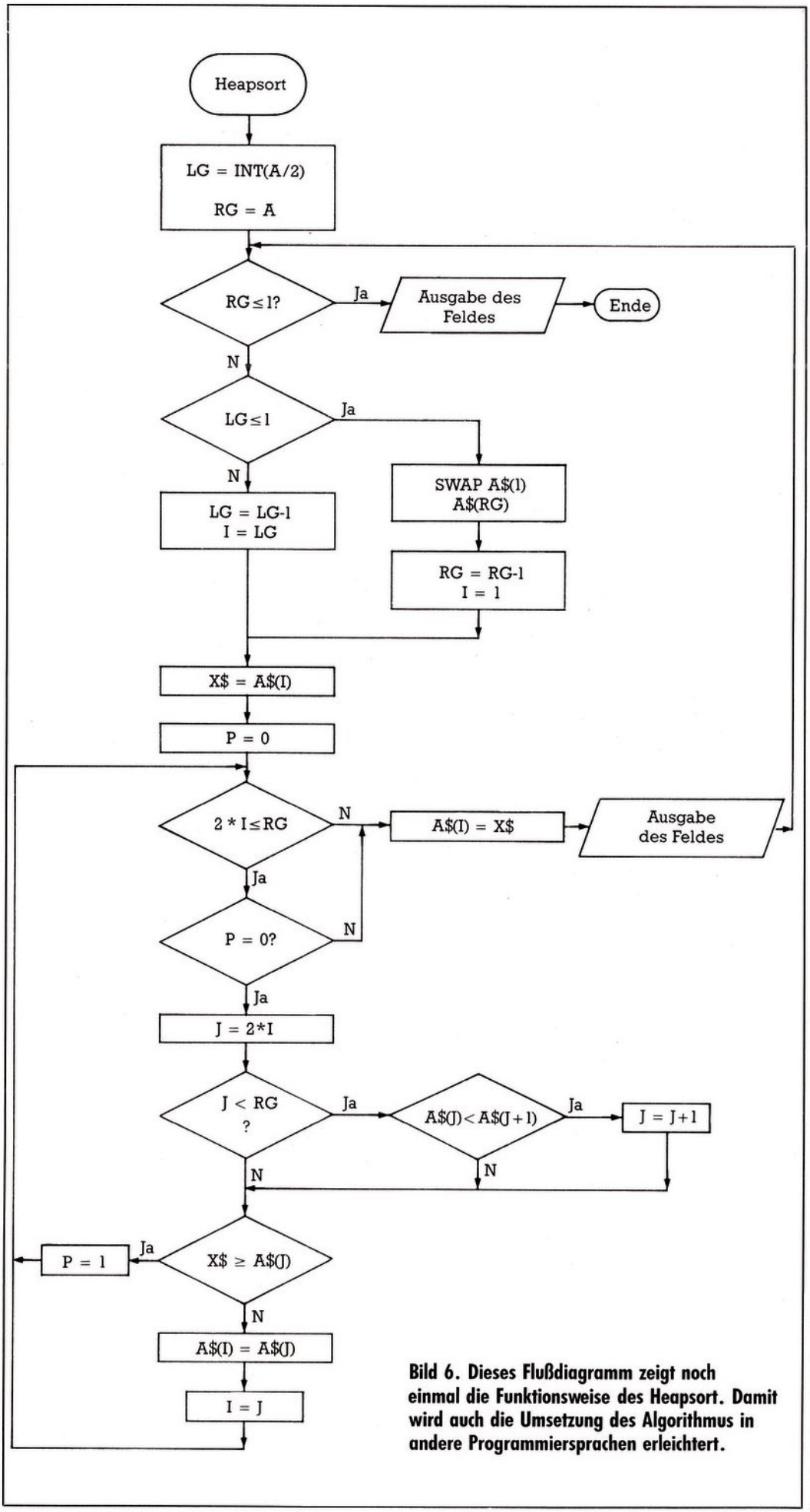

Bild 6. Dieses Flußdiagramm zeigt noch einmal die Funktionsweise des Heapsort. Damit wird auch die Umsetzung des Algorithmus in andere Programmiersprachen erleichtert.

Fortsetzung von Seite 115

Einführung in CAD mit dem Commodore C 64

Das rechnerunterstützte Konstruieren (Computer aided Design, kurz CAD) macht Gebrauch von den Fähigkeiten des Computers, zu rechnen, Daten zu verwalten und (auf dem Bildschirm oder dem Grafikdrukker) zu zeichnen, womit hier technisches Zeichnen, nicht Malen gemeint ist. Der Autor dieses neuen Data-Becker-Buches betont zu Beginn, daß im Vergleich zu professionellen Systemen die geringe Bildschirm-Auflösungsfähigkeit des C 64 der einzige echte Nachteil sei; die geringe Speicherkapazität und Rechengeschwindigkeit sei durch geschickte Organisation und Geduld wettzumachen. Und so stellt er Schritt für Schritt eine Reihe von kleinen Bausteinen zusammen (Zeichnen von Punkt, Linie, Rechteck, Kreis, Bogen; Strichlinien, Maßpfeile, Schraffuren), aus denen wiederum Makroelemente (Quader, Pyramide, Prisma, Zylinder, Kegel, Kegelstumpf, Kugel) gebildet werden können.

Techniken wie Vergrößerung, Verkleinerung, Zoomen, Duplizieren, Drehen, Spiegeln und Setzen oder Löschen von Einzelpunkten mit einem »Hires-Cursor« werden ebenfalls behandelt und gehören sicherlich zu den interessantesten und nützlichsten Ausführungen des Buches. So hat man schließlich einen Satz von Bausteinen mit den entsprechenden Programmen in Simons Basic zur Verfügung, mit denen man eigene Aufgaben angehen kann.

In diesem Zusammenhang ist allerdings anzumerken, daß die Lösung der betrachteten Probleme hauptsächlich in den Programmen selbst stattfindet und im Text relativ wenig erklärt wird. Dazu gesellt sich die Bemerkung: »Wer wissen will, wie ein Programm strukturiert ist, kommt auch ohne Erklärungen dahinter, und wer es ohne Erklärungen nicht schafft, wird auch mit ihnen nicht glücklich.« Anstelle von 35 Seiten Programmlisting für eine spezielle Beispielzeichnung wäre es vielleicht angebrachter, einige grundsätzliche Dinge etwas ausführlicher zu erklären; seine Programme für eigene Anwendungen muß der Leser schließlich doch selber schreiben.

Auf dem Wege dazu bietet dieses Buch letztlich aber doch eine Reihe von nützlichen Hilfsmitteln, Anregungen und Lösungsansätzen, so daß man das Buch jedem an technischer Grafik Interessierten durchaus empfehlen kann. (Dr. Wolfgang Bunse)

Info: Werner Heift, Einführung in CAD mit dem Commodore 64, Data Becker, Düsseldorf 1985, 302 Seiten, ISBN 3-89011-067-3, 49 Mark.